Зміст

[1. Кроки для старту сервісу 1](#_Toc88057794)

[2. Кроки для запуску тестів 1](#_Toc88057795)

[3. Обрана методологія/алгоритм 2](#_Toc88057796)

[4. Опис роботи алгоритму 3](#_Toc88057797)

[5. Опис основних інтерфейсів/класів 4](#_Toc88057798)

[IBoxCutting (BoxCrossCutting, BoxCrossRotatedCutting and etc.) 4](#_Toc88057799)

[IBoxCuttingPattern (BoxTetrisCuttingPattern) 5](#_Toc88057800)

[IPacker (BinaryTreePacker) 5](#_Toc88057801)

[ICommand (StartCommand, StopCommand, UpCommand and etc) 5](#_Toc88057802)

[BoxCutter 5](#_Toc88057803)

[6. Опис розкріїв коробки 6](#_Toc88057804)

[BoxCrossCutting 7](#_Toc88057805)

[BoxCrossRotatedCutting 7](#_Toc88057806)

[BoxTetrisCuttingPattern 7](#_Toc88057807)

[7. Ідеї для зменшення кількості відходів 8](#_Toc88057808)

[8. Ідеї для збільшення швидкодії алгоритму 8](#_Toc88057809)

[9. Ідеї для масштабування сервісу 9](#_Toc88057810)

## 1. Кроки для старту сервісу

В папці з файлом docker-compose.yml виконати команду:

docker-compose up

## 2. Кроки для запуску тестів

Див BoxCutting.UnitTests проект.

Для Linux чи MacOs

docker run --rm -v ${pwd}:/app -w /app mcr.microsoft.com/dotnet/sdk:3.1 dotnet test

Для Windows Command Line (Linux контейнери)

docker run --rm -v %cd%:/app -w /app mcr.microsoft.com/dotnet/sdk:3.1 dotnet test --logger:trx

Для Windows PowerShell (Linux контейнери)

docker run --rm -v ${pwd}:/app -w /app mcr.microsoft.com/dotnet/sdk:3.1 dotnet test --logger:trx

Для Windows PowerShell (Windows контейнери)

docker run --rm -v ${pwd}:C:\app -w C:\app mcr.microsoft.com/dotnet/sdk:3.1 dotnet test --logger:trx

Очікуваний output (займає трохи часу, бо білдить проект в контейнері)

Test Run Successful.

Total tests: 12

Passed: 12

Total time: 2.6907 Seconds

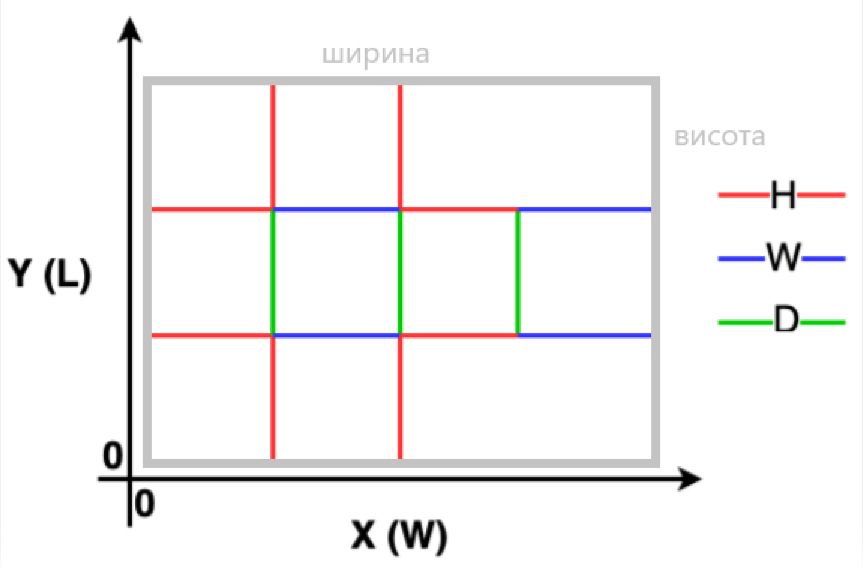
## 3. Обрана методологія/алгоритм

Задача зводиться до того, щоб розмістити максимальну кількість розкріїв коробки на заданому листі картону.

Розкрій коробки являє собою фігуру випуклої форми (може бути багато варіантів розкрію коробки).

Кожен розкрій коробки можна описати прямокутником, який вміщає в себе цей розкрій. На картинці нижче цей прямокутник має розміри

,



Тоді ефективність конкретного розкрію можна оцінити по формулі

В найкращому випадку Ефективність = 1. В залежності від розмірів коробки ефективність розкрію буде різна.

В результаті задача спрощується до того, що потрібно розмістити максимальну кількість прямокутників розкріїв на листі картону.

Таке спрощення суттєво полегшує імплементацію алгоритму.

Так як задача лежить в площині NP-hard проблем, я вибрав FirstFit Heuristic алгоритм. А в імплементації алгоритму використовується Binary Tree структура даних.

Основні ідеї, які були реалізовані:

1. Алгоритм передбачає можливість додавання різних розкріїв коробки, а також патернів розкрію (див IBoxCuttingPattern інтерфейс).
2. Алгоритм завжди намагається розмістити спочатку найбільш ефективний розкрій коробки на листі картону (стільки разів скільки це можливо), якщо це не можливо, тоді береться наступний по ефективності розкрій.
3. Алгоритм сортує розкрії по спаданню ширини або висоти в залежності від розмірів листа картону (щоб досягнути максимальної ефективності розміщення)
4. Алгоритм використовує BinaryTree, щоб зберігати вільні області, куди можна розмістити розкрій. Та використовує рекурсію, для знаходження вільної області для розміщення розкрію коробки.

Переваги:

* Простота алгоритму (завдяки тому, що використовується єврестичний підхід)
* Розширюваність, можливість покращувати результати шляхом додавання більш ефективних розкріїв коробки або паттетнів розкрію
* Швидкодія, завдяки тому, що не потрібно перебирати всі можливі варіанти розміщення коробок та розкріїв

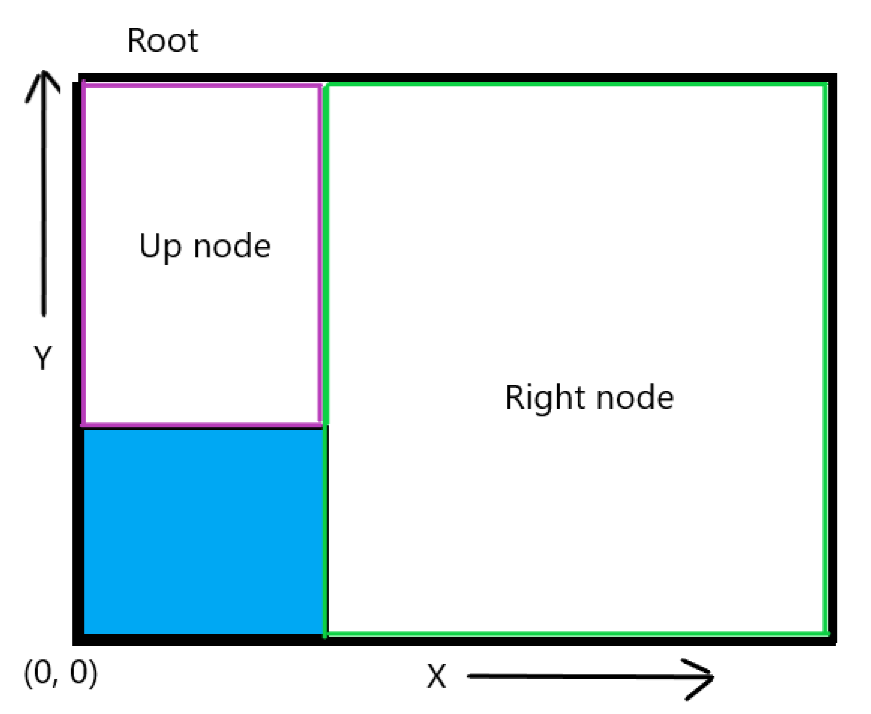
Недоліки

* Не завжди оптимальні результати
* Необхідність шукати та програмувати ефективні розкрії коробки, оскільки для різних розмірів коробки розкрії будуть мати різну ефективність
* Не враховується можливість повертати розкрій
* Необхідність добавляти повернуті варіанти розкріїв для покращення результатів

## 4. Опис роботи алгоритму

Вхідними даними для алгоритму є Sheet (лист картону) та List<IBoxCutting> (список розкріїв коробки).

Реалізація алгоритму представлена в класі BinaryTreePacker.cs. Кожний елемент бінарного дерева представлений класом Node, в якому X, Y, With, Height – координати і розміри вільного простору на листі картону. Right, Up – гілки дерева (вільні області праворуч та вгорі), Used – властивість, яка вказує на те чи в ноду вже було розміщено рокзрій.

1. Ініціалізовується Root нода бінарного дерева, яка має розміри такі самі як розміри листа картону.
2. Рокзрії коробок сортуються по ефективності, а потім по спаданню висоти або ширини (в залежності від розмірів листа)
3. Починаючи з найбільш ефективного розкрію, в залежності чи розкрій має паттерн
   1. Якщо розкрій має паттерн, то береться найближче вільне місце в дереві, і паттерн будує відповідний розкрій коробок для даного кусочка листа картону.
   2. Якщо розкрій немає паттерну, то починається рекурсивний пошук вільного місця в дереві для розмірів даного розкрію.
4. Якщо нода знайдена (або паттерн побудований), вона відмічається як використана та відбувається Split цієї ноди (квадрат розбивається на 2 менших квадрати, праворуч та вгорі, див. малюнок) 
5. Далі, алгоритм пробує знайти місце для того самого розкрію (або паттерну розкрію), до тих пір, поки не залишиться місце для даного розкрію.
6. Якщо для розкрію невдається найти вільне місце на листу, береться наступний розкрій з списку
7. Все продовжується до поки не буде знайдено місце для найбільш нефективного розкрію (\* тутє ідея для оптимізації)

## 5. Опис основних інтерфейсів/класів

### IBoxCutting (BoxCrossCutting, BoxCrossRotatedCutting and etc.)

Інтерфейс, який описує характеристики розкрію коробки

1. **int BoxCount –** кількість коробок, які можна скласти з даного розкрію. Імплементація передбачає, що в системі можуть бути розкрії, з яких можна скласти декілька коробок (це може значно знизити кількість відходів)
2. **int Width –** довжина прямокутника, який вміщає розкрій коробки
3. **int Height –** висота прямокутника, який вміщає розкрій коробки
4. **double Efficiency –** ефективність розкрію
5. **IReadOnlyList<ICommand> GetProgram(Point initialPoint) –** метод, який повертає список команд для станка, потрібних для того, щоб вирізати розкрій. **initialPoint** – це точка на листі картону, відносно якого розміщений прямокутник розкрію. Відповідно метод **GetProgram** вертає інструкції, які будуть вирізати розкрій з прямокутника (відносно точки на листі картону, де цей прямокутник розміщений)

### IBoxCuttingPattern (BoxTetrisCuttingPattern)

Інтерфейс, який наслідується від IBoxCutting, має ті самі характеристики розкрію коробки, а також метод Build , який дозволяє масштабувати паттерн на заданий лист картону

1. **IBoxCutting Build(Sheet sheet) –** метод, який вертає розкрій для вхідного листа картону (по паттерну, який запрограмований в GetProgram методі розкрію). Модель Sheet – описує якусь область на вхідному листі картону.
2. **BoxCuttingPatternType Type –** тип паттерну, якщо HeightBased – значить паттер складає рокзрії вверх (тобто фіксована ширина, а висота може бути різна в залежності від висоти листа картону). Якщо WidthBased – значить паттер складає розкрії вправо (тобто висота фіксована, а ширина може бути різна в залежності від ширини листа картону)

### IPacker (BinaryTreePacker)

Інтерфейс, який представляє логіку розміщення рокзріїв коробки на листі картону.

List<BoxPackingResult> PackBoxCuttings(Sheet sheet, List<IBoxCutting> boxCuttings)

PackBoxCuttings – метод який приймає лист картону, та список розкріїв коробки. Повертає список розкріїв та їхнє розміщення на листі картону.

### ICommand (StartCommand, StopCommand, UpCommand and etc)

Інтерфейс, який представляє команду станка.

### BoxCutter

Основний клас, основні цілі якого:

1. Валідація вхідних данних
2. Створення всеможливих розкріїв коробки IBoxCutting через IBoxCuttingFactory інтерфейс (SOLID принципи… + краща абстракція для написання юніттестів + моків)
3. Виклик IPacker.PackBoxCuttings , який розміщує розкрії на листі картону
4. Агрегація команд для станка з кожного розкрію коробки, який був розміщений на листі картону
5. Агрегація кількості коробок з кожного розкрію

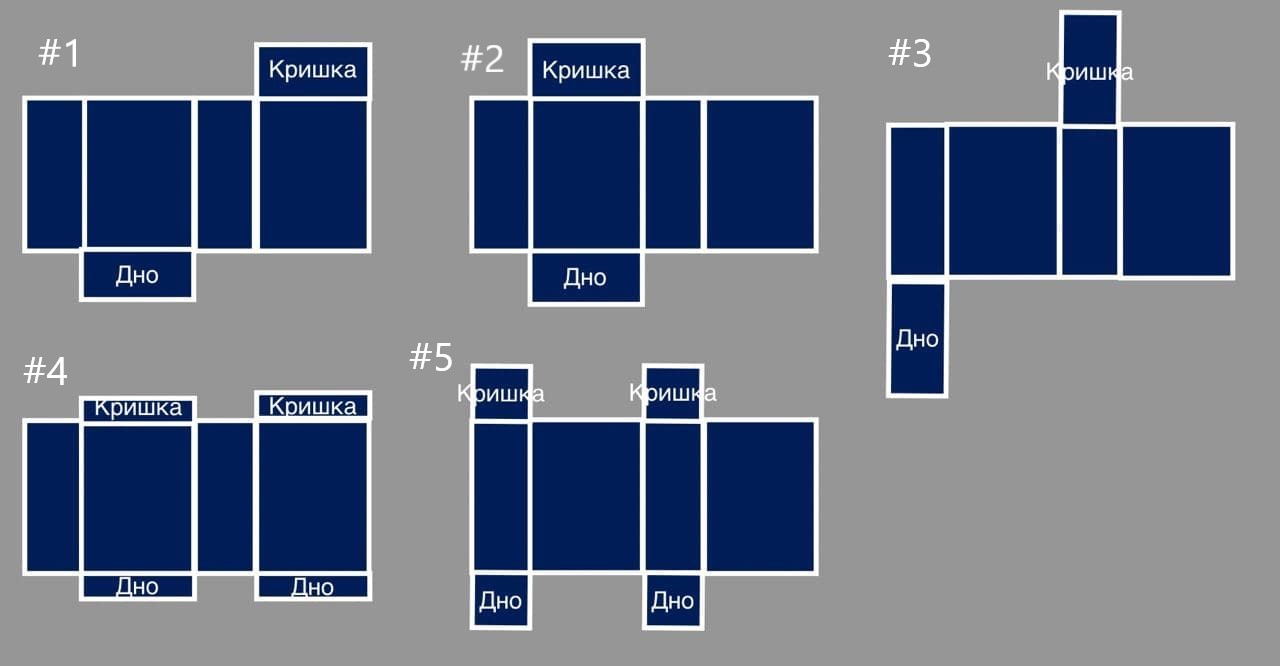
## 6. Опис розкріїв коробки

Так як алгоритм передбачає можливість добавлення будь-якої кількості розкріїв коробки (і паттернів), моя ціль була імплементувати основні розкрії, які є ефективними (згідно з формулою, яка була наведена спочатку)

Я не встиг реалізувати більш складніші та ефективніші розкрії і паттерни розкріїв через відстуність часу☹

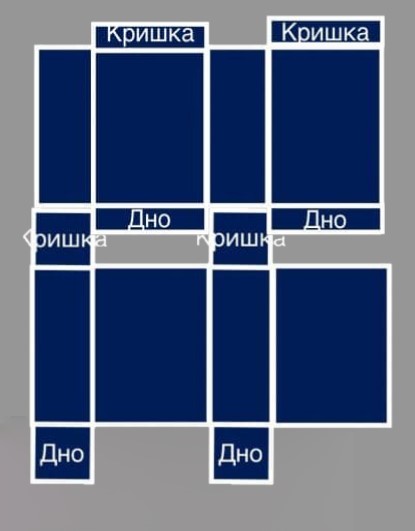
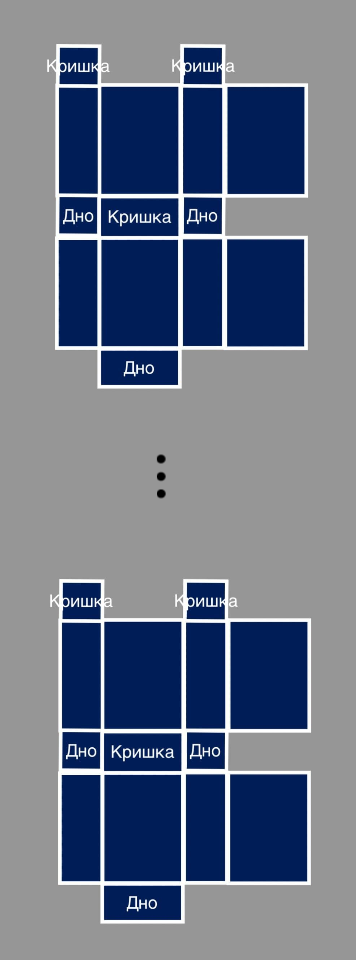
Але, в теорії, сервіс можна покращувати 😊

На картинці нижче наведені прості розкрії (оскільки сторони коробки можуть бути різними, то і наведені розкрії будуть мати різну ефективність, тому потрібно їх всіх імплементувати, щоб мати гарну ефективність при різних розмірах)



Малюнок 6.1, варіанти простих розкріїв

На малюнку нижче наведені приклади складних розкріїв (та паттернів розкрію), вони є завжди ефективнішими ніж прості розкрії

Малюнок 6.2, варіанти паттернів розкрію коробки

### BoxCrossCutting

Імплементація простого розкрію №2 (див малюнок 6.1)

### BoxCrossRotatedCutting

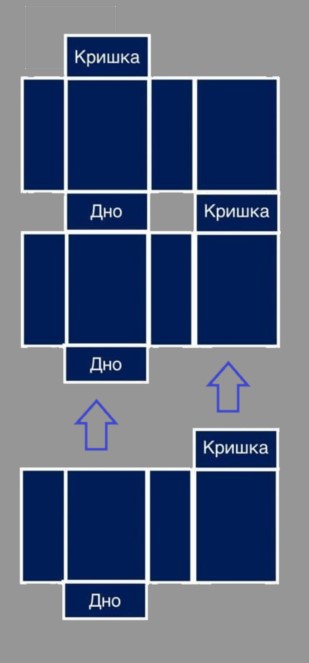
Імплементація простого розкрію №2, повернутого на 90 градусів за годинниковою стрілкою (див малюнок 6.1)

BoxHalfCrossCutting

Імплементація простого розкрію №1 (див малюнок 6.1)

### BoxTetrisCuttingPattern

Імпрементація паттерну розкрію, який використовує BoxCrossCutting та BoxHalfCrossCutting розкрії, та розміщує їх так, щоб ефективно використати місце на картоні. Див. Малюнок 6.3



Малюнок 6.3, паттерн розкрію коробки (скомпонований з простого розкрію №1 та розкрію №2)

Імплементація цього паттерну будує команди розкрію №1, потім пересуває різак в на висоту цього розкрію, і далі будує команди розкрію №2.

Це повторюється RepentanceFactor разів.

Тут я не дореалізував можливість повторювати паттер частоково, тобто паттер зараз повторюється повністю (незнаю як це пояснити краще)

## 7. Ідеї для зменшення кількості відходів

Для покращення роботи програми, необхідно придумати більше та більш ефективні патерни розкріїв коробки. Це можуть бути розкрії, на яких розміщені 2 або більше розкріїв таким чином, що ефективніше використовується простір листа картону.

Також потрібно реалізувати як найбільше різниї варіантів розкрію, так як при різних розмірах коробки, вони будуть мати різну ефективність.

В ідеалі треба придумати розкрій, який б мав ефективність як найближче до 1. Зрозуміло, що такий розкрій буде в себе вміщати багато коробок, і буде мати великі розміри. Але завдяки тому, що Алгоритм спочатку намагається вставити найбільш ефективний розкрій, ми отримаємо меншу кількість відходів.

## 8. Ідеї для збільшення швидкодії алгоритму

Для збільшення швидкодії алгоритму в мене є декулька ідей:

* Додати більше паттернів розкріїв коробки, тоді час розміщення розкріїв на велищезночу листі картону зменшиться (оскільки алгоритм буде оперувати великими блоками і масштабуватись для будь-якого розміру листа, відповідно буде менше ітерацій пошуку місця в дереві).
* Розпаралелити пошук вільного місця в BinaryTree, зараз алгоритм шукає місце спочатку в правій Right Node , а потім Up Node.
* Замінити рекурсію на список вільних блоків (це зменшить використання RAM памяті)
* При великій кількості розкріїв, треба буде додати умову виходу з пошуку, якщо в BinaryTree не залишилось ніодної Node, де б теоритично міг би вміститись розкрій
* Кешування результатів
  + Система може кешувати вже пораховані результати, і формувати новий IBoxCuttingPattern на їх основі, тоді цей паттерн можна буде використати для білших листів картону (що в рази збільшить швидкодію)

## 9. Ідеї для масштабування сервісу

Оскільки в сервісу немає якогось Local State, а по суті сервіс є Stateless, то ми легко можемо масштабувати кількість сервісів, і розподіляти нагрузку на них Лоад Балансером.

1. Лоад Баланс стратерія може бути Round Robin. У випадку, якщо буде реалізований кеш, то можна використати Distributed Cache, щоб він був спільним для всіх інстансів сервісу.
2. Для спрощення можна використати Лоад Баланс стратегію Sticky Session (так як один станок скоріше за все буде декілька раз запитувати розкрій для одного й того самого листа картону). В кожному сервісі кешувати результати в оперативну пам'ять. Цей варіант буде простішим, але недоліком буде те, що в кожного сервісу буде свій кеш, який може містити однакові дані.

